BENCHMARKS SUBSISTEMA BEOWULF

Linpack

En el test Linpack realizado en Julio de 2002, el resultado obtenido fue de 10,3 GFlops. El benchmark ha sido compilado con los compiladores de Portland versión 3.3-2, GM 1.5.1_Linux y MPICH-GM 1.2.1..7b.

Pallas MPI Benchmark PMB

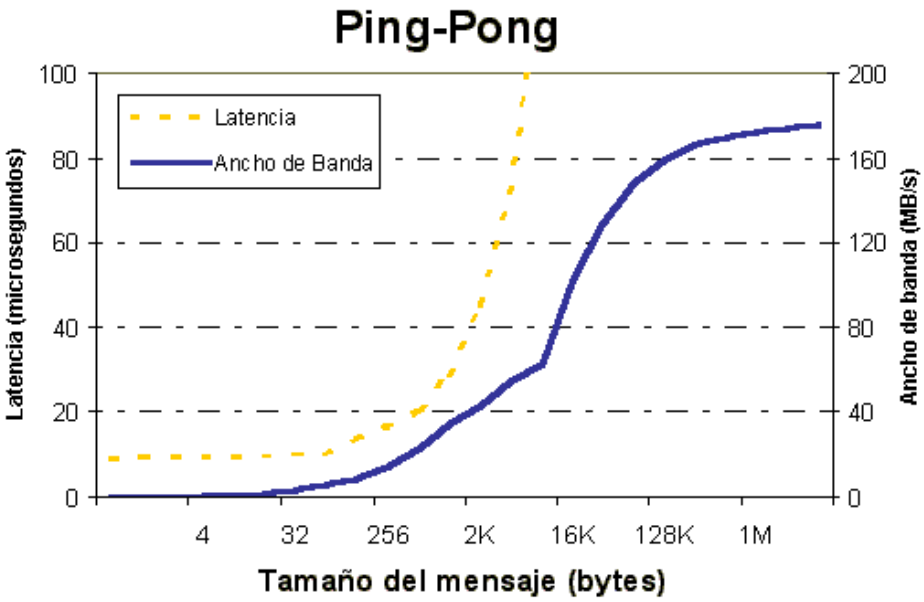
A continuación se recogen los resultados obtenidos de los benchmarks de Pallas GmbH. Estos benchmarks se emplean para comparar el rendimiendo de las implementaciones MPI. Los resultados se han obtenido en Julio del 2002 con la versión GM 1.5.1_Linux y la versión MPICH-GM 1.2.1..7b, basada en MPICH 1.2.1. El benchmark PMB ha sido compilado con los compiladores de Portland pgcc, versión 3.3-2.

Rendimiento Punto a Punto

El rendimiento punto a punto se mide entre dos procesos que se ejecutan en dos nodos distintos, y se expresan en MBytes por segundo de ancho de banda por nodo (enviar + recibir), así como la latencia en las comunicaciones expresada en microsegundos.

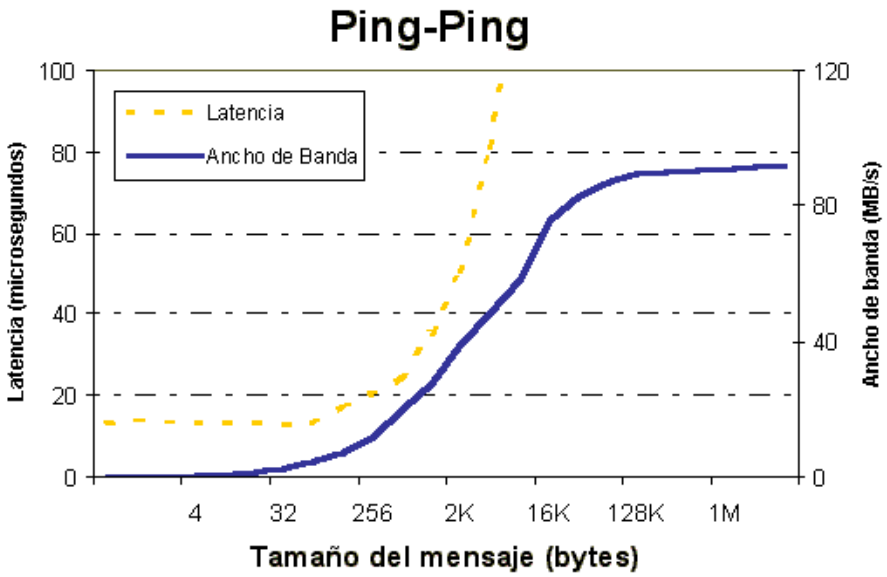
PMB PingPong

PingPong es el patrón clásico utilizado para medir el arranque y el throughput de un único mensaje enviado entre dos nodos. La secuencia de comunicación es un bucle MPI_Recv() seguida por un MPI_Send().



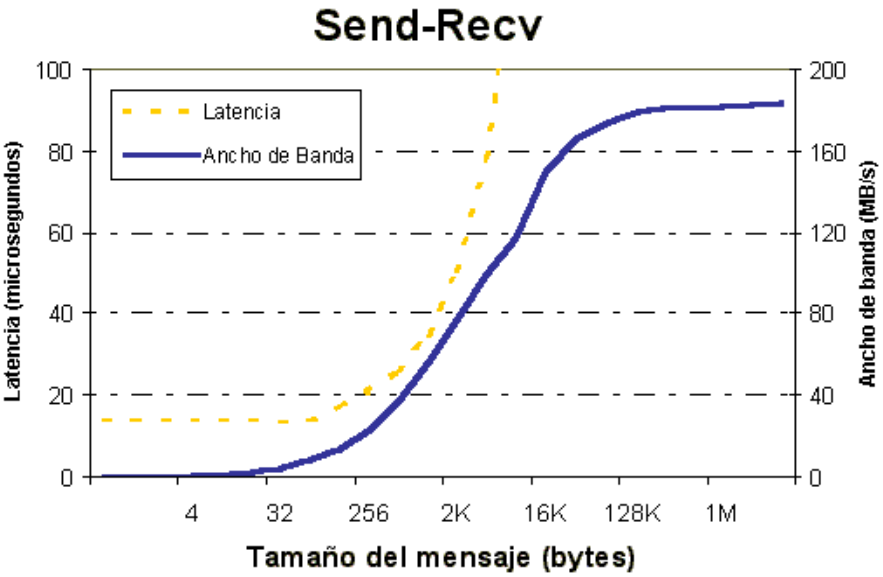
PMB PingPing

PingPing es un test de comunicación en dos direcciones concurrentes. Al igual que PingPong, PingPing mide el arranque y el throughput de un único mensaje enviado entre dos procesos, con la diferencia de que los mensajes están obstruidos por mensajes en la dirección contraria. Para ello, dos procesos se comunican (MPI_Isend/MPI_Recv/MPI_Wait) entre sí, con llamadas MPI_Isend enviadas simultáneamente.



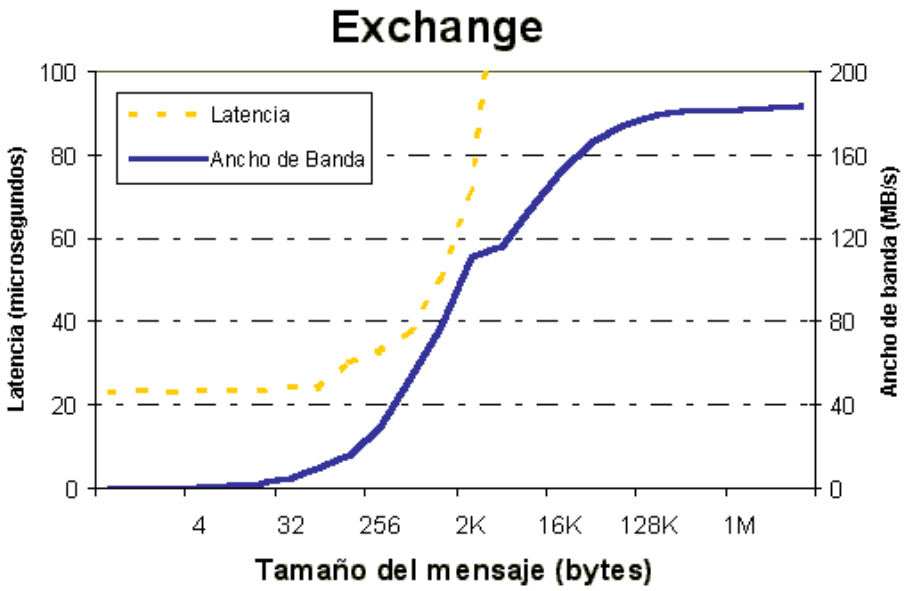
PMB SendRecv

Este test está basado en la llamada MPI_Sendrecv(). En él los procesos forma un cadena de comunicación periódica, en la cual cada proceso envía al vecino que se encuentra a su derecha y recive del vecino que se encuentra a su izquierda en la cadena.



PMB Exchange

En este test, el grupo de procesos también actúa como una cadena periódica, y cada proceso intercambia (exchanges) datos con sus vecinos derecho e izquierdo en la cadena.

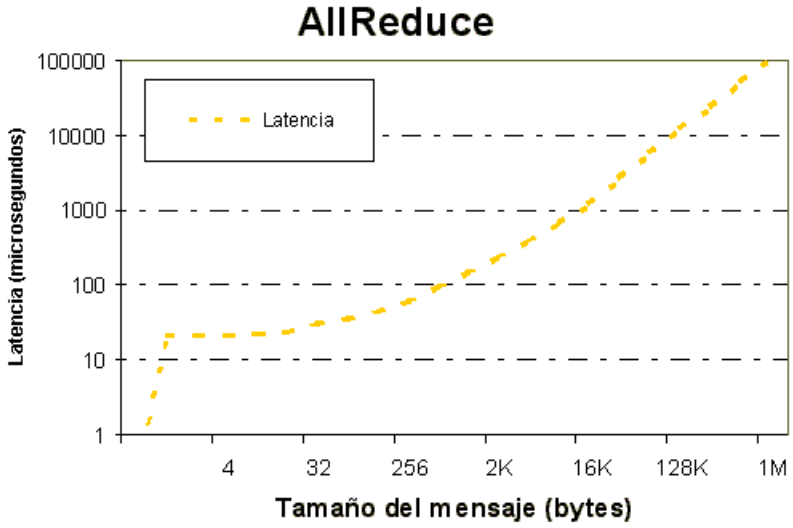


Benchmarks colectivos

El rendimiendo colectivo o de la interconexión del sistema en su conjunto, se mide entre todos o un subconjunto de los nodos del sistema. Los datos de los benchmarks colectivos muestran la latencia media en las comunicaciones en microsegundos.

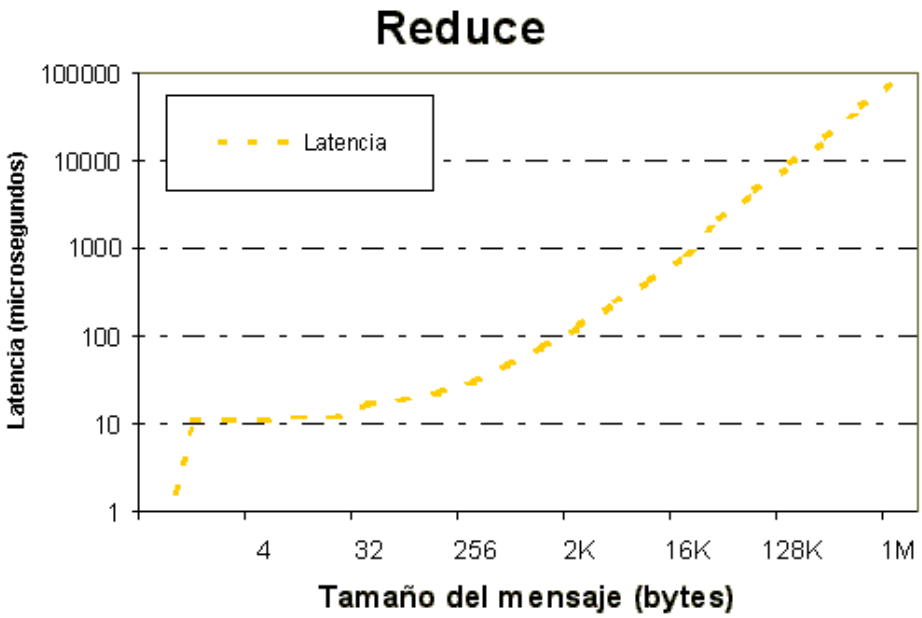
PMB Allreduce

Este es el benchmark de la funcion MPI_Allreduce. Reduce vectores de números en punto flotante de longitud L = X/sizeof(float) desde cada proceso a un único vector y lo distribuye a todos los procesos. El tipo de datos MPI es MPI_FLOAT y la operación MPI es MPI_SUM.



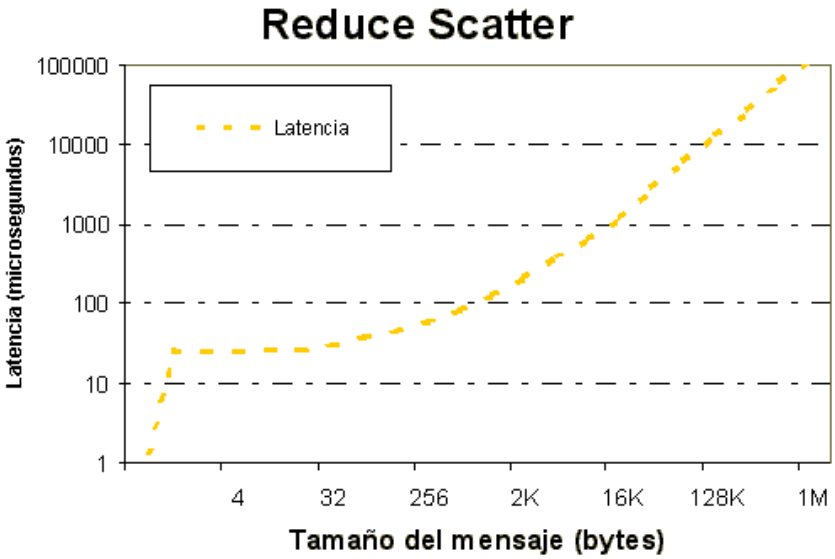
PMB Reduce

Este es el benchmark de la función MPI_Reduce. Reduce vectores de números en punto flotante de longitud L = X/sizeof(float) desde cada proceso a un único vector en el proceso padre. El tipo de dato MPI es MPI_FLOAT y la operación MPI es MPI_SUM. El proceso padre de la operación cambia cíclicamente.



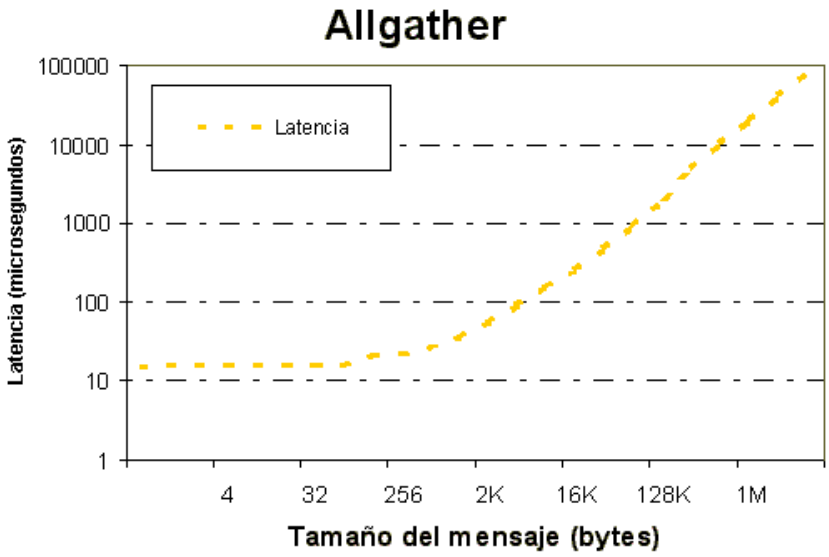
PMB Reduce_scatter

Este es el benchmark de la función MPI_Reduce_scatter. Reduce vectores de números en punto flotante de longitud L = X/sizeof(float) en un único vector. El tipo de dato MPI es MPI_FLOAT y la operación MPI es MPI_SUM. En la fase dispersa (scatter), los valores de L se dividen uniformemente entre todos los procesos.



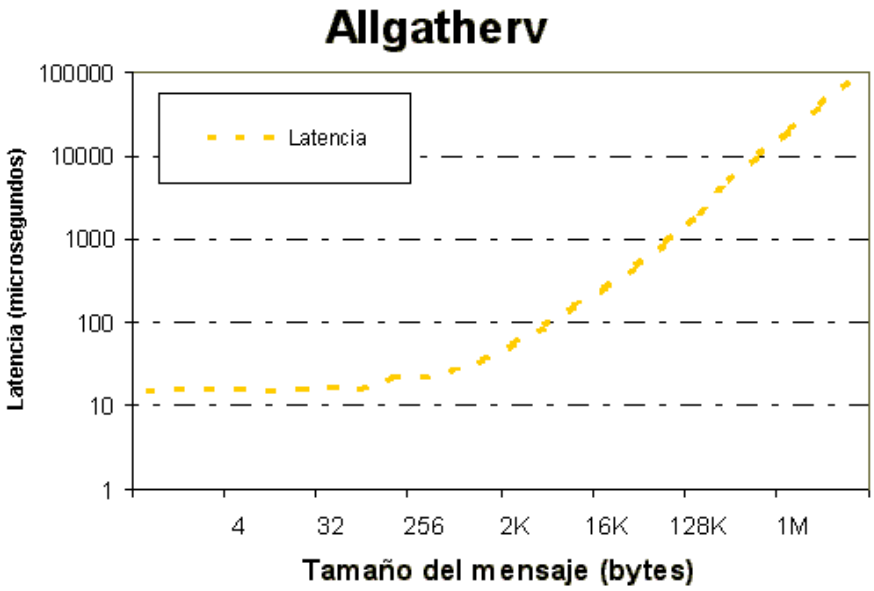
PMB Allgather

Este es el benchmark de la función MPI_Allgather. Cada proceso envía X bytes y recibe el grupo de los X*(nº_procesos) bytes.



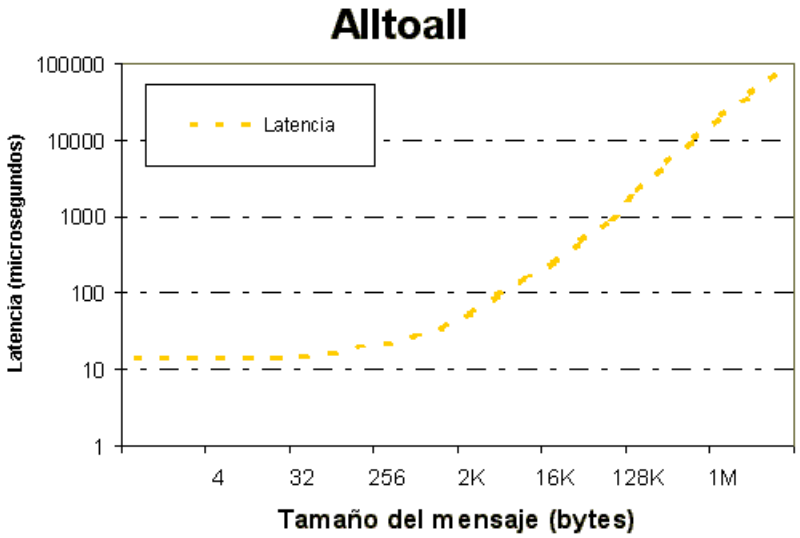
PMB Allgatherv

Este es el benchmark de la función MPI_Allgatherv. Cada proceso envía X bytes y recibe el grupo de los X*(nº_procesos) bytes.



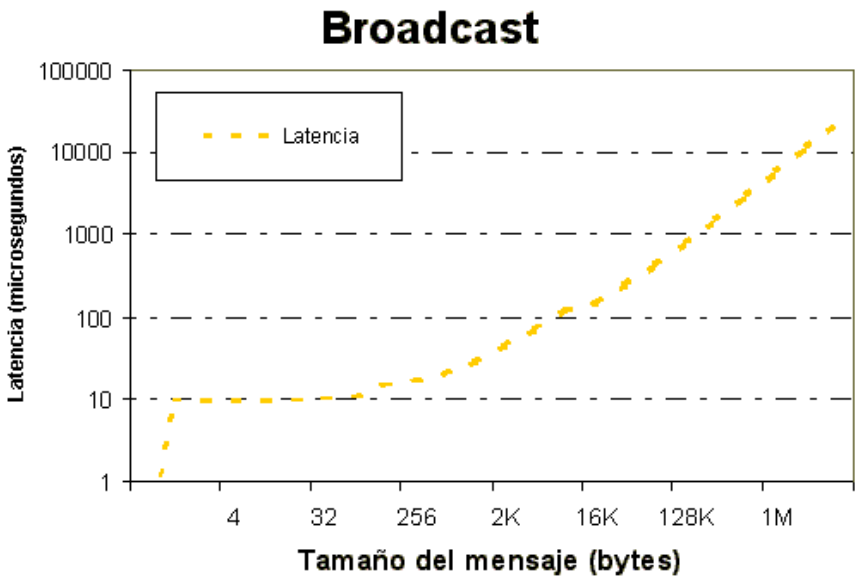
PMB Alltoall

Este es el benchmark de la función MPI_Alltoall. Cada proceso envía y recibe X*(nº_procesos) bytes (X para cada proceso).



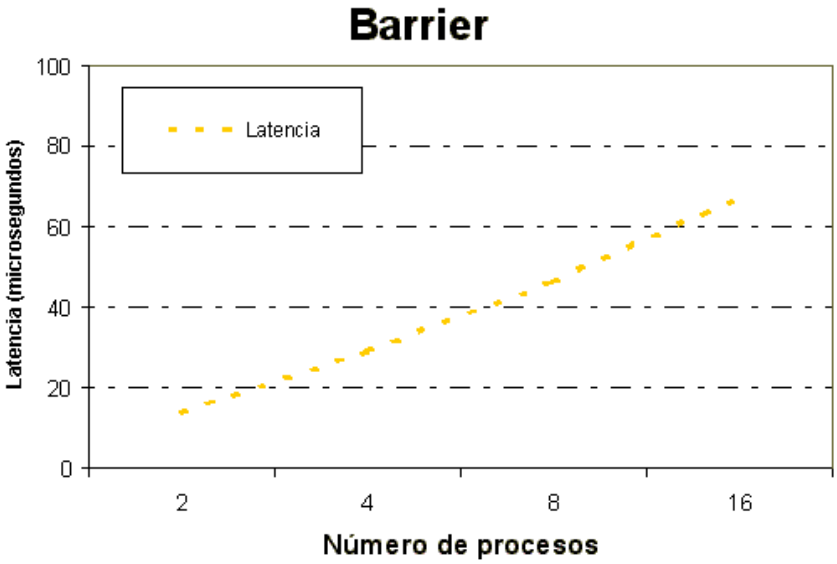
PMB Broadcast

Este es el benchmark de la función MPI_Bcast. Un proceso padre envía (broadcasts) X bytes a todos los otros procesos.



PMB Barrier

Este es el benchmark de la función MPI_Barrier(). No se intercambia ningún dato.

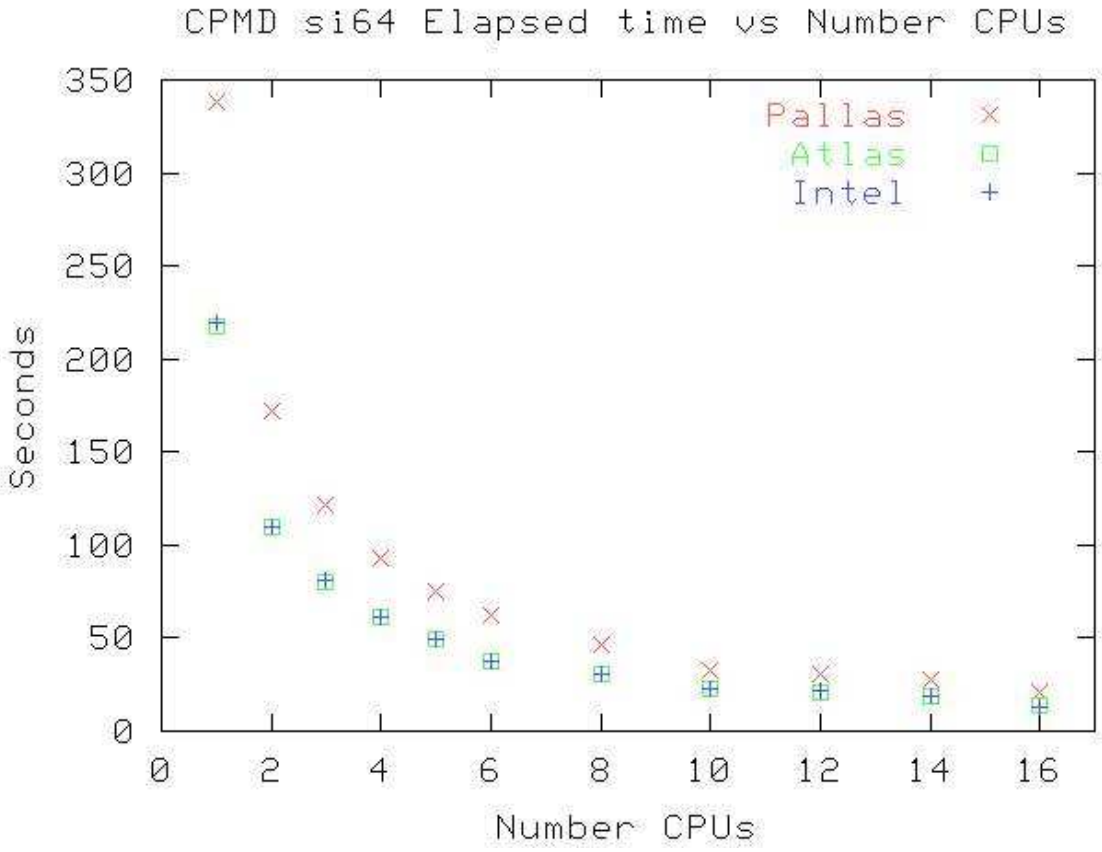


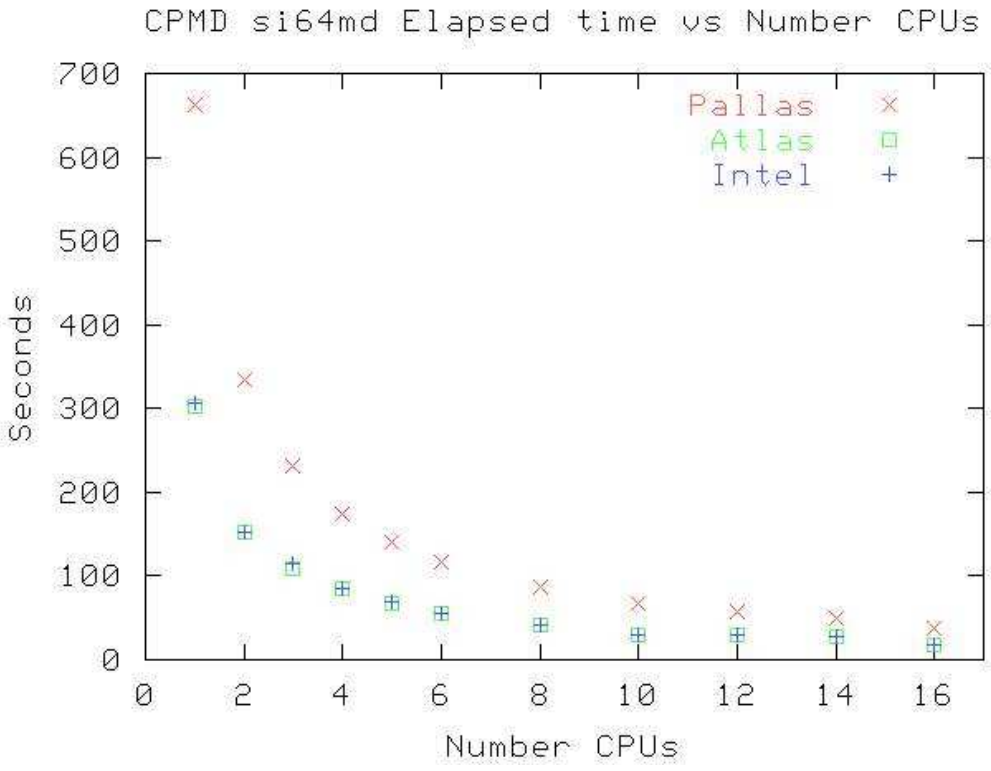
CPMD Si64

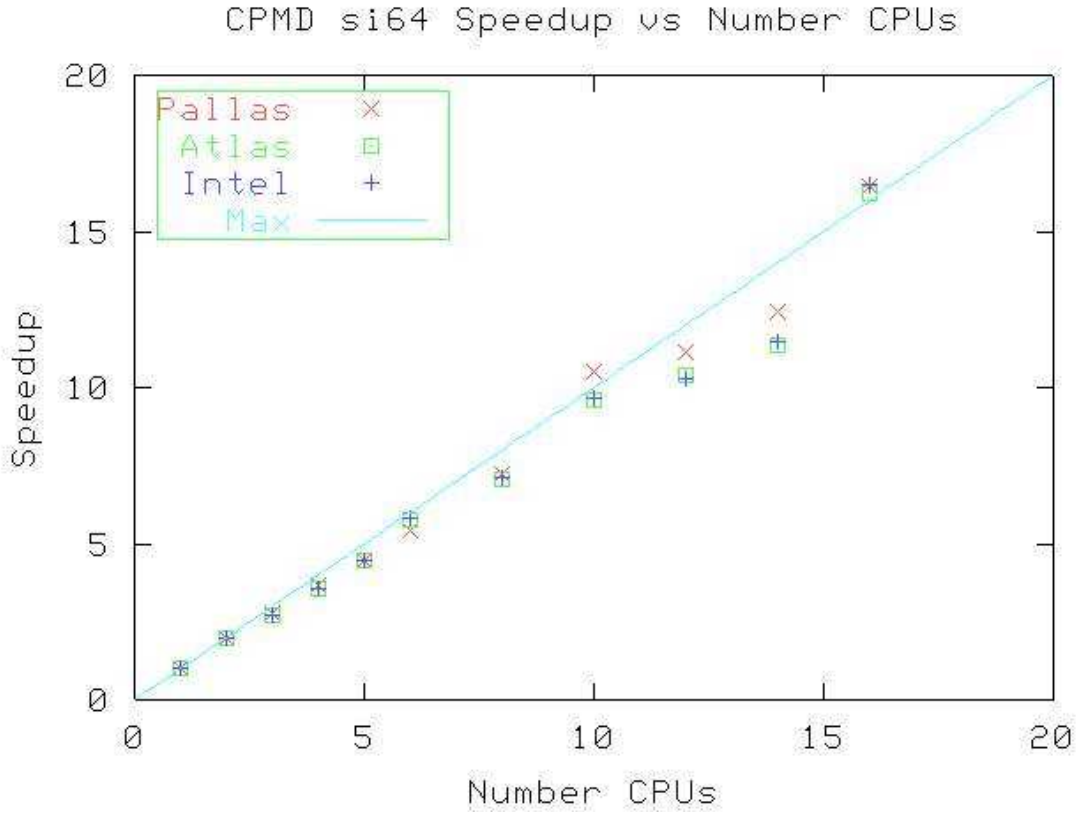
Para probar BeoWulf con una aplicación real, se compilo el programa de dinámica molecular CPMD en este sistema utilizando MPI. El compilador utilizado fue el de Portland y se utilizaron tres librerías diferentes de BLAS:

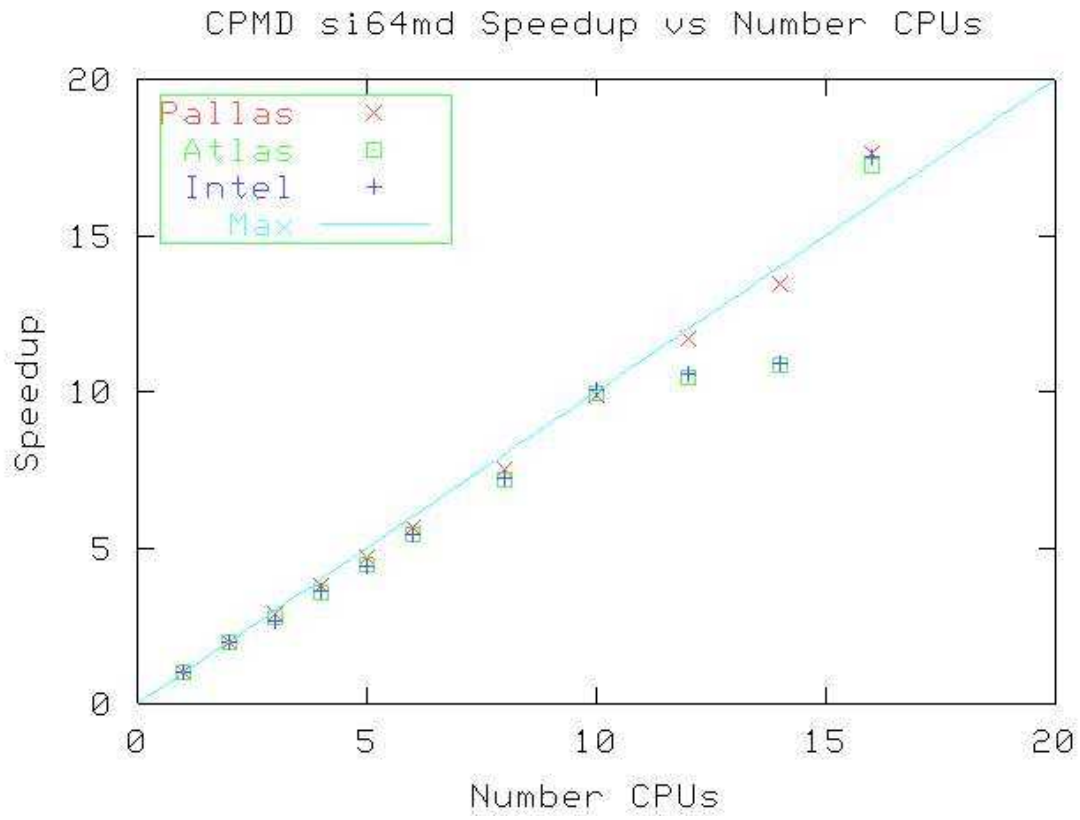
- Las propias del compilador de Portlan (marcadas en las figuras como Pallas)
- Las del proyecto Atlas
- Las propias del fabricante (Intel)

Como entrada se utilizo un cojunto de 64 moléculas de Si sobre lo que se hizo una optimización de la función de onda (marcado como Si64 en las figuras) y, un segundo paso, un cálculo ad initio MD (marcado como Si64md). En las figuras siguientes se muestran los resultados obtenidos tanto en tiempo consumido como en SpeedUp en función del número de procesadores utilizados.


CPMD si64 Elapsed time vs Number CPUs
Pallas
Atlas
Intel
Seconds
Number CPUs
0
50
100
150
200
250
300
350
0
2
4
6
8
10
12
14
16


CPMD si64md Elapsed time vs Number CPUs
Pallas
Atlas
Intel
Seconds
Number CPUs
0
100
200
300
400
500
600
700
0
2
4
6
8
10
12
14
16


CPMD si64 Speedup vs Number CPUs
Pallas
Atlas
Intel
Max
Speedup
Number CPUs
0
5
10
15
20


CPMD si64md Speedup vs Number CPUs
Pallas
Atlas
Intel
Max
Speedup
Number CPUs
0
5
10
15
20